In the Specification:

Please amend the Specification as shown below. Applicant respectfully submits that the proposed amendments correct informalities in the Specification and that no new matter is being added.

Please replace paragraph **[0003]** with that shown below:

[0003] The ~~J2EE~~ JAVA 2 ENTERPRISE EDITION (J2EE) specification defines the ~~Enterprise Archive~~ ENTERPRISE ARCHIVE (EAR) file as a standard structure for developing and packaging J2EE applications. EARs are useful for application development, in that the application may, for example, include both a Web application (Webapp) and an ~~Enterprise Java Bean~~ ENTERPRISE JAVABEAN (EJB), which will be packaged into the EAR. However, while this works well for completed applications, it isn't very convenient for application development.

Please replace paragraph **[0005]** with that shown below:

[0005] To deploy the application, a number of steps must be performed, namely compiling the ~~Java~~ JAVA files, generating any servlets or container classes, and packaging the whole lot in an EAR. The EAR adheres to a format wherein the top level includes an EAR descriptor /META-INF/application.xml, and all of the other components are listed underneath. This approach works well when the application has been fully completed and ready for installation in the production environment. However it's less useful for application development.

Please replace paragraph **[0007.1]** with that shown below:

[0007.1] The terms JAVA, ~~JAVA 2 Enterprise Edition~~ JAVA 2 ENTERPRISE EDITION (J2EE), ~~Enterprise Java Bean~~ ENTERPRISE JAVA BEAN (EJB), and ~~Java servlet Page~~ JAVA

SERVLET PAGE (JSP), are trademarks of Sun Microsystems, Inc. The terms ~~WebLogic~~ WEBLOGIC, and ~~WebLogic Server~~ WEBLOGIC SERVER, are trademarks of BEA Systems, Inc.

Please replace paragraph **[0010]** with that shown below:

[0010] An embodiment of the invention provides a development-oriented directory structure that can be used with an application server (for example the ~~WebLogic Server~~ WEBLOGIC SERVER from BEA Systems, Inc.), and which solves a number of the problems associated with traditional ~~Enterprise Archive~~ ENTERPRISE ARCHIVE (EAR) files. The development-oriented directory structure avoids the copying of files during the build process, and presents a clean separation between human-readable source files stored in a source control system and generated ~~Java~~ JAVA class files. The two directories (source and output) appear like a single application.

Please replace paragraph **[0014]** with that shown below:

[0014] The split-directory system provides a recommended directory layout and an accompanying set of ant tasks for building and deploying applications. The split-directory differs from traditional EAR files because it is optimized for iterative development. Instead of a single archived EAR file or an "exploded" EAR directory structure, the split-directory has 2 parallel directories. The source directory contains all of the source files e.g. ~~Java~~ JAVA files, descriptors, JSP files, HTML etc in a traditional EAR-like directory structure. All generated or compiled classes are stored in a separate output directory. This arrangement produces several advantages over the traditional exploded EAR file:

1. No need for file copying.
2. Web files can be changed and redeployed in place without rebuilding.
3. Clean separation between source and generated files allows cleaning the build by just removing the output directory.

Please replace paragraph **[0018]** with that shown below:

[0018] wlcompile is the main build task which compiles the application's ~~Java~~ JAVA files. wlcompile begins by analyzing all of the components and determining their type. It treats components as either EJBs, Web applications, or ~~Java~~ JAVA components. ~~Java~~ JAVA components are compiled first into the output directory/APP-INF/classes so they will be visible throughout the application. Next, wlcompile builds the EJBs and automatically includes the previously built ~~Java~~ JAVA components in the compiler's classpath. This allows the EJB components to call the ~~Java~~ JAVA components without requiring the user to manually edit their classpath. Finally the .java files in the webapp are compiled with the EJB components and ~~Java~~ JAVA components in the compiler's classpath. This allows the Web Applications to refer to the EJB and application ~~Java~~ JAVA classes without manually editing the classpath. The following example compiles the application with sources in /acme/myapp to output directory /build/myapp:

```
<wlcompile srcdir="/acme/myapp" destdir="/build/myapp" />
```

Please replace paragraph **[0023]** with that shown below:

[0023] The ejbapp source is contained within a directory (AcmeEJB). As usual, the descriptors are in the META-INF directory. Any ~~Java~~ JAVA files under the AcmeEJB root directory are compiled into the output directory.

Application with webapp and ejb

```
InventoryApp/META-INF/application.xml
InventoryApp/META-INF/weblogic-application.xml

InventoryApp/AcmeEJB/META-INF/ejb-jar.xml
InventoryApp/AcmeEJB/META-INF/weblogic-ejb-jar.xml
InventoryApp/AcmeEJB/com/acme/ejb/MyLocal.java
InventoryApp/AcmeEJB/com/acme/ejb/MyHome.java
InventoryApp/AcmeEJB/com/acme/ejb/MyEJB.java

InventoryApp/AcmeWeb/WEB-INF/web.xml
InventoryApp/AcmeWeb/WEB-INF/weblogic.xml
```

```
InventoryApp/AcmeWeb/WEB-INF/src/com/acme/web/MyServlet.java
InventoryApp/AcmeWeb/login.jsp
InventoryApp/AcmeWeb/graphics/logo.jpg
```

Please replace paragraph **[0026]** with that shown below:

[0026] This example demonstrates how the Inventory Application can be extended to use ~~Java~~ JAVA utility classes. ~~Java~~ JAVA Utility classes differ from third-party jars because the source files are part of the application and must be compiled. ~~Java~~ JAVA Utility classes are typically libraries used by application components such as EJBs or webapps. The build process compiles these files into the APP-INF/classes directory under the output directory. These classes are avaliable throughout the application.